Chapter 2

Machine Language

Machine language

- The only language a computer can understand directly.
- Each type of computer has its own unique machine language.
- We will study the machine language of the H1 computer.

H1

- Simplified model of a modern computer
- Easy to learn
- Simulated by the **sim** program in the H1 Software Package. **sim** runs on DOS, Windows, X86 Linux, Sun SPARC Solaris, and Macintosh OS X.

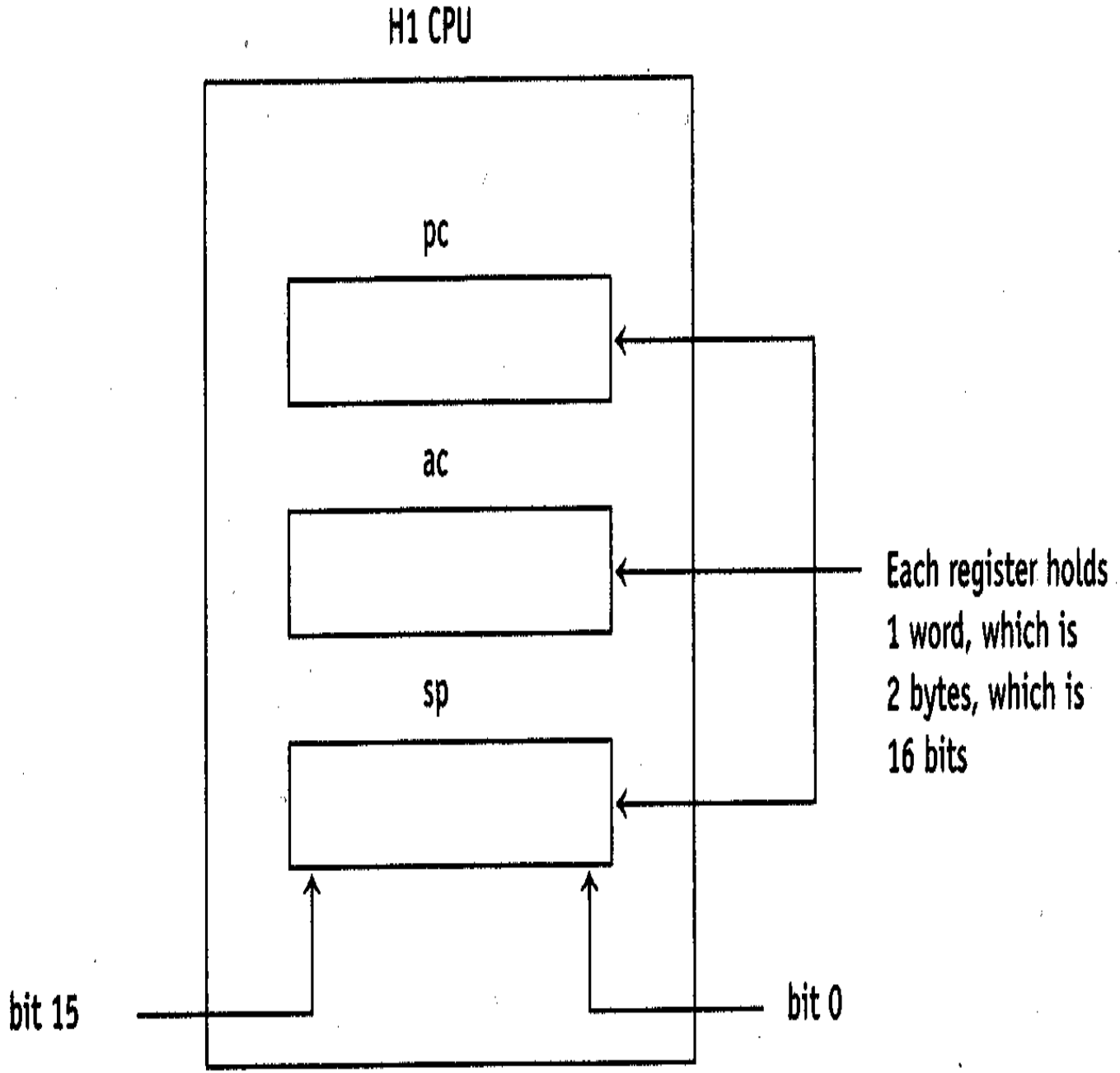
Central Processing Unit (CPU) of H1

- The heart of H1 is its central processing unit (CPU).
- The CPU contains computational circuits, control circuits, and registers.
- A *register* is a special memory area that can hold one 16-bit number.

Registers on H1

- Program counter (pc)
- Accumulator (ac)
- Stack pointer (sp)

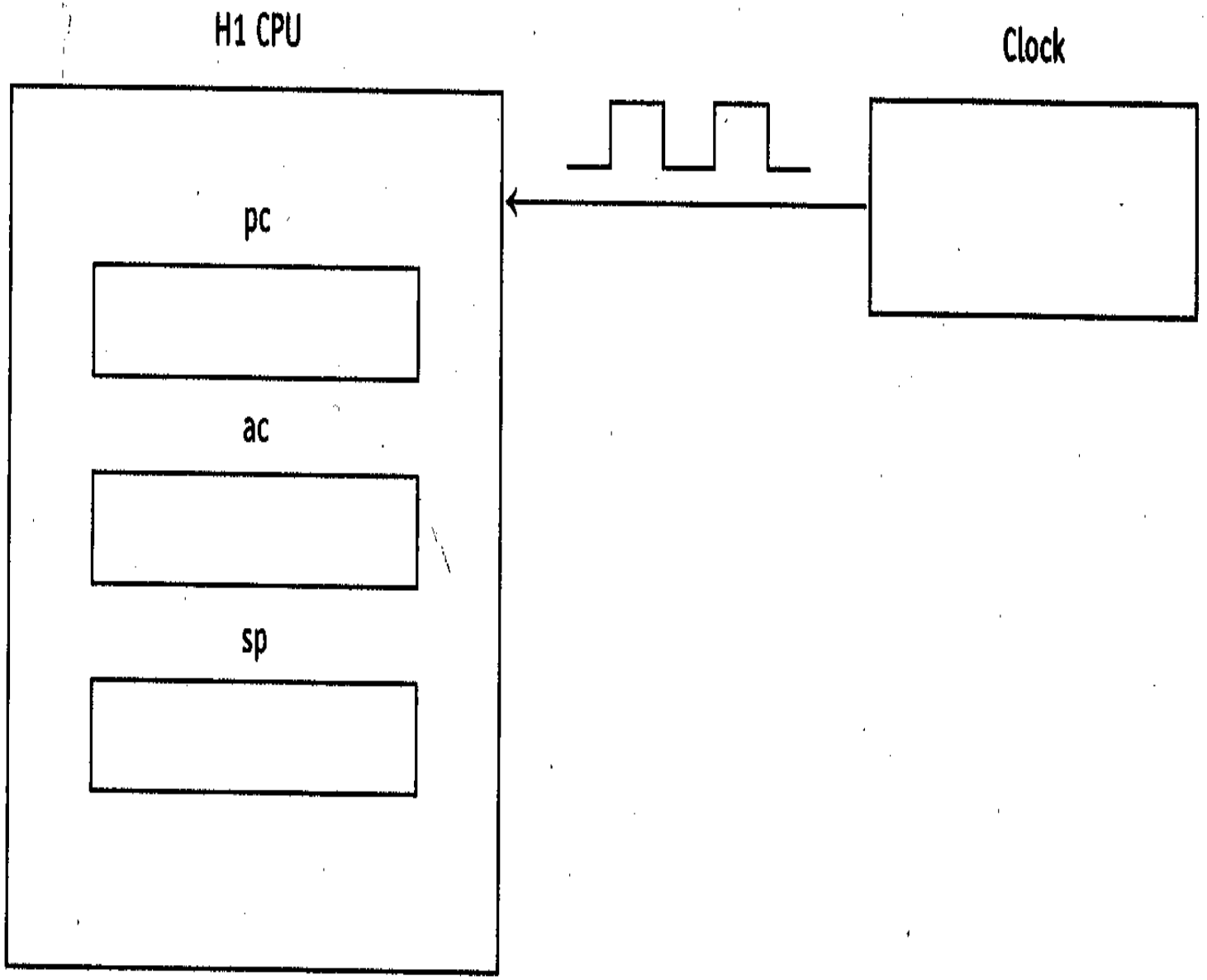
FIGURE 2.1


H1 CPU
pc
ac
sp
Each register holds
1 word, which is
2 bytes, which is
16 bits
bit 15
bit 0

Clock

- Generates stream of pulses that determines the speed at which the CPU performs its operation.
- One Hertz is 1 pulse per second.
- One megaHertz is 1 million pulses per second.
- One gigaHertz is 1 billion pulses per second.

FIGURE 2.2



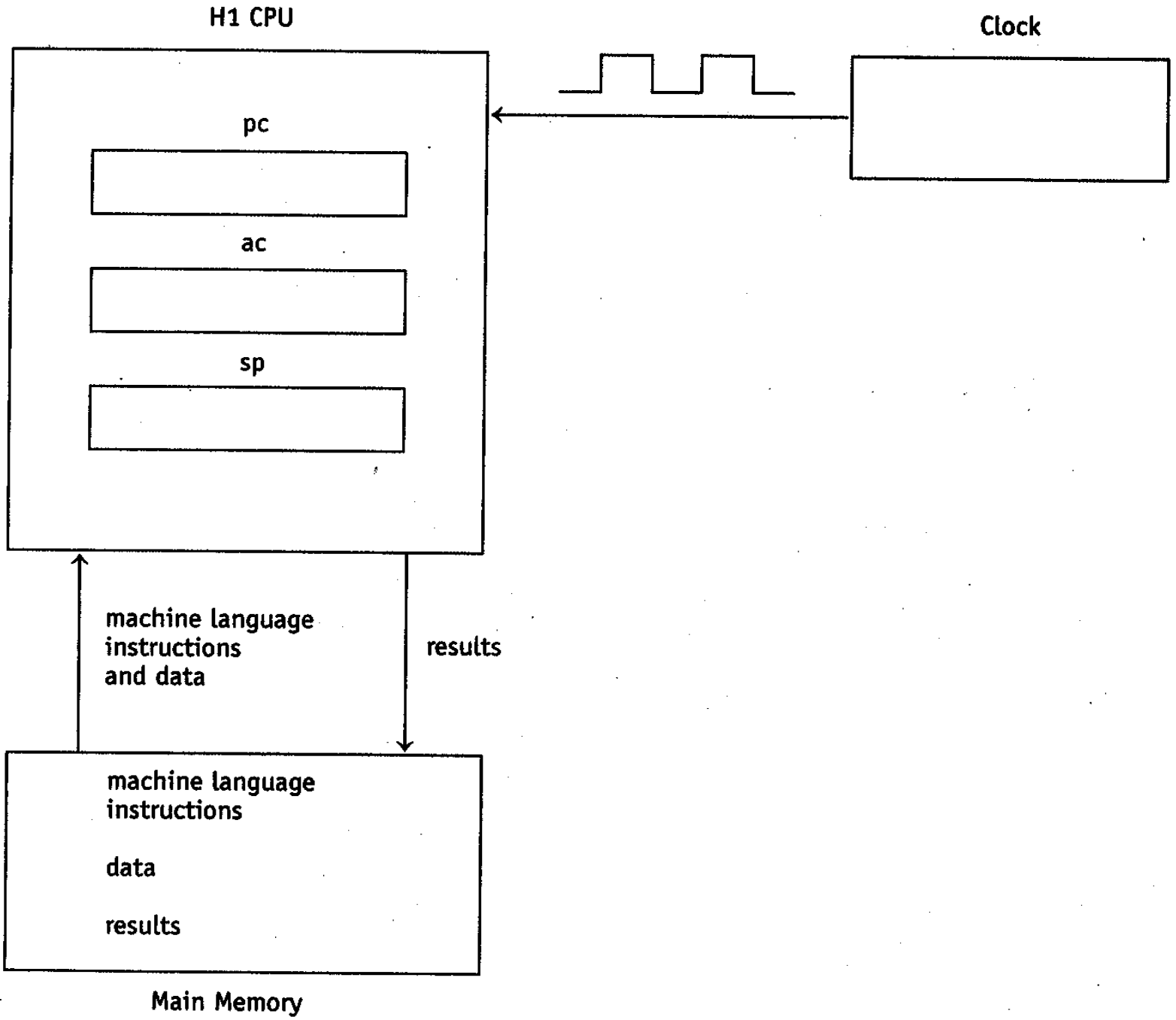
Machine language instructions

- Binary numbers
- Tell the CPU what to do
- The only kind of instruction the CPU can execute
- The CPU does NOT understand C++ or Java instructions. Instructions in these languages have to be translated to machine language.

Main memory

- Holds machine instructions, data, and results.
- CPU gets data and machine instructions from main memory.
- CPU places results into main memory.

FIGURE 2.3


H1 CPU
Clock
pc
ac
sp
machine language
instructions
and data
results
machine language
instructions
data
results
Main Memory

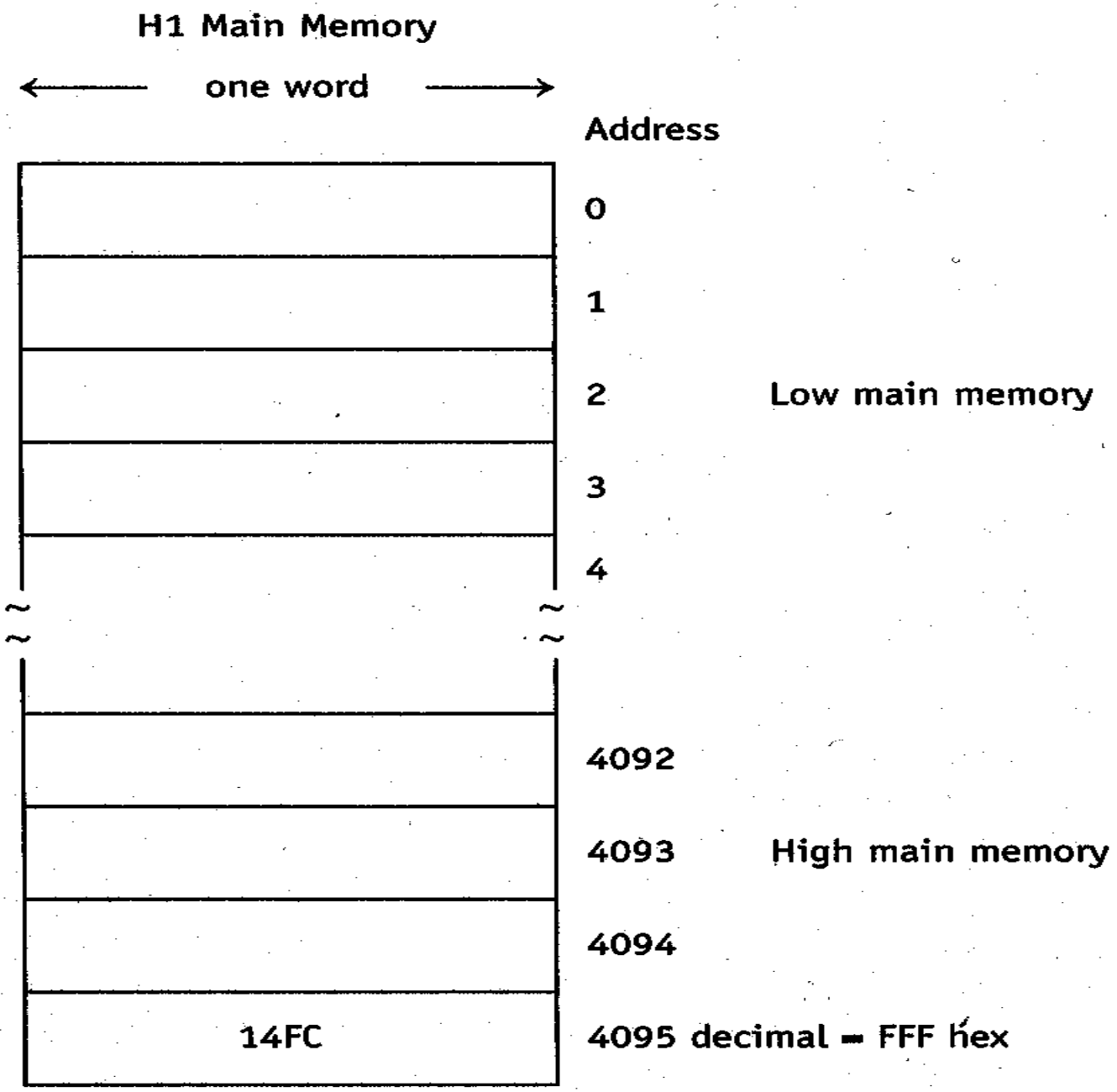
Why computers are fast

- The CPU and main memory are all electronic. They have NO moving parts to slow them down.
- As long as main memory can provide the CPU with machine instructions and data fast enough, the CPU can perform computations at an enormous rate.

Organization of main memory

- An array of memory slots
- Each slot is one *word* (16 bits) on the H1. On most computers, each slot is one *byte* (8 bits).
- Each slot has an identifying number called its *address*.
- Addresses are sequential integers starting from 0.
- What is in a memory slot is called its *contents.*

FIGURE 2.4


H1 Main Memory
one word
Address
0
1
2
3
4
4092
4093
4094
14FC
4095 decimal = FFF hex
Low main memory
High main memory

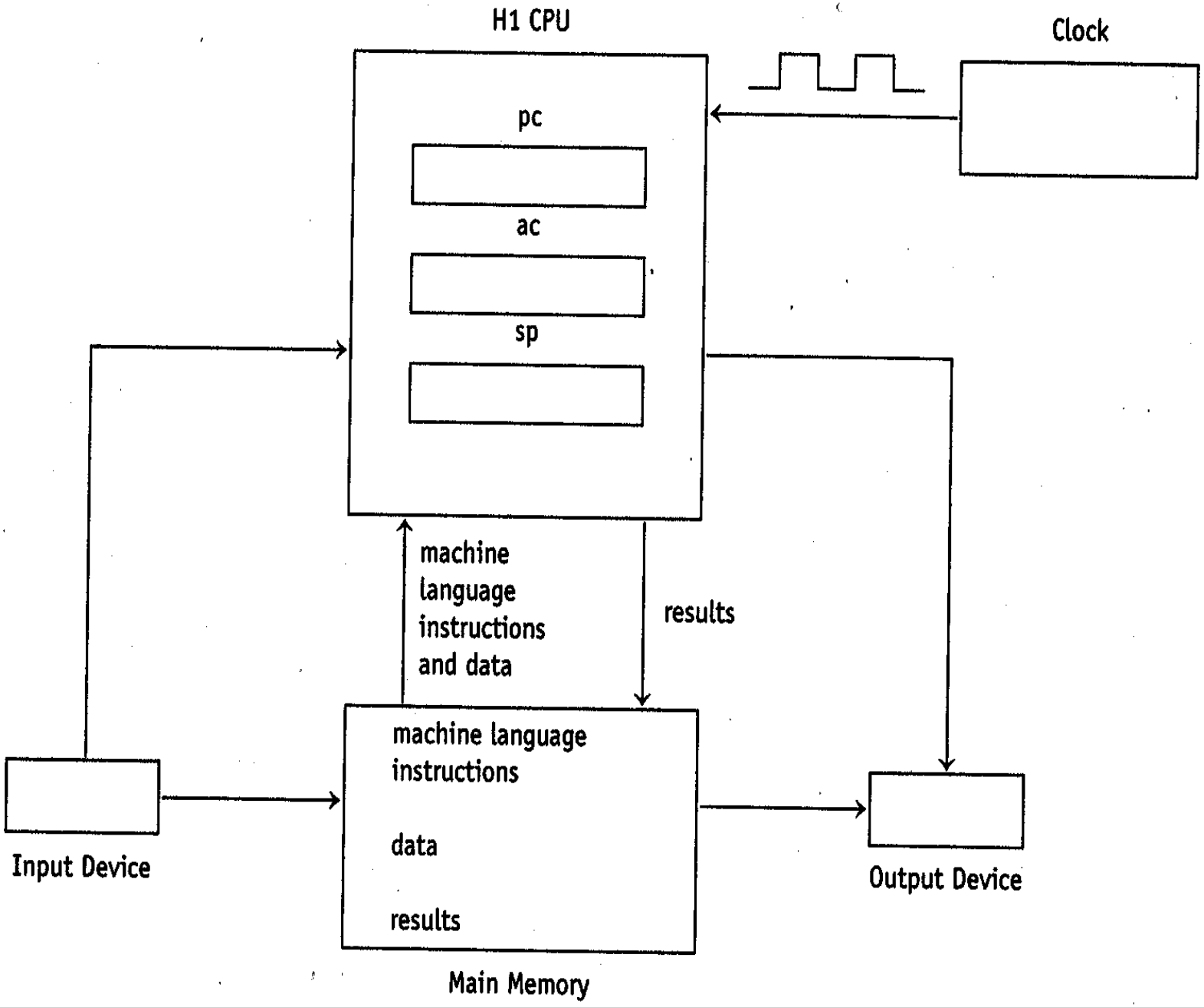
Be sure to distinguish between the address of a main memory slot and its contents.

The contents is the binary number *in* the slot. The address is the number that identifies that slot.

I/O devices

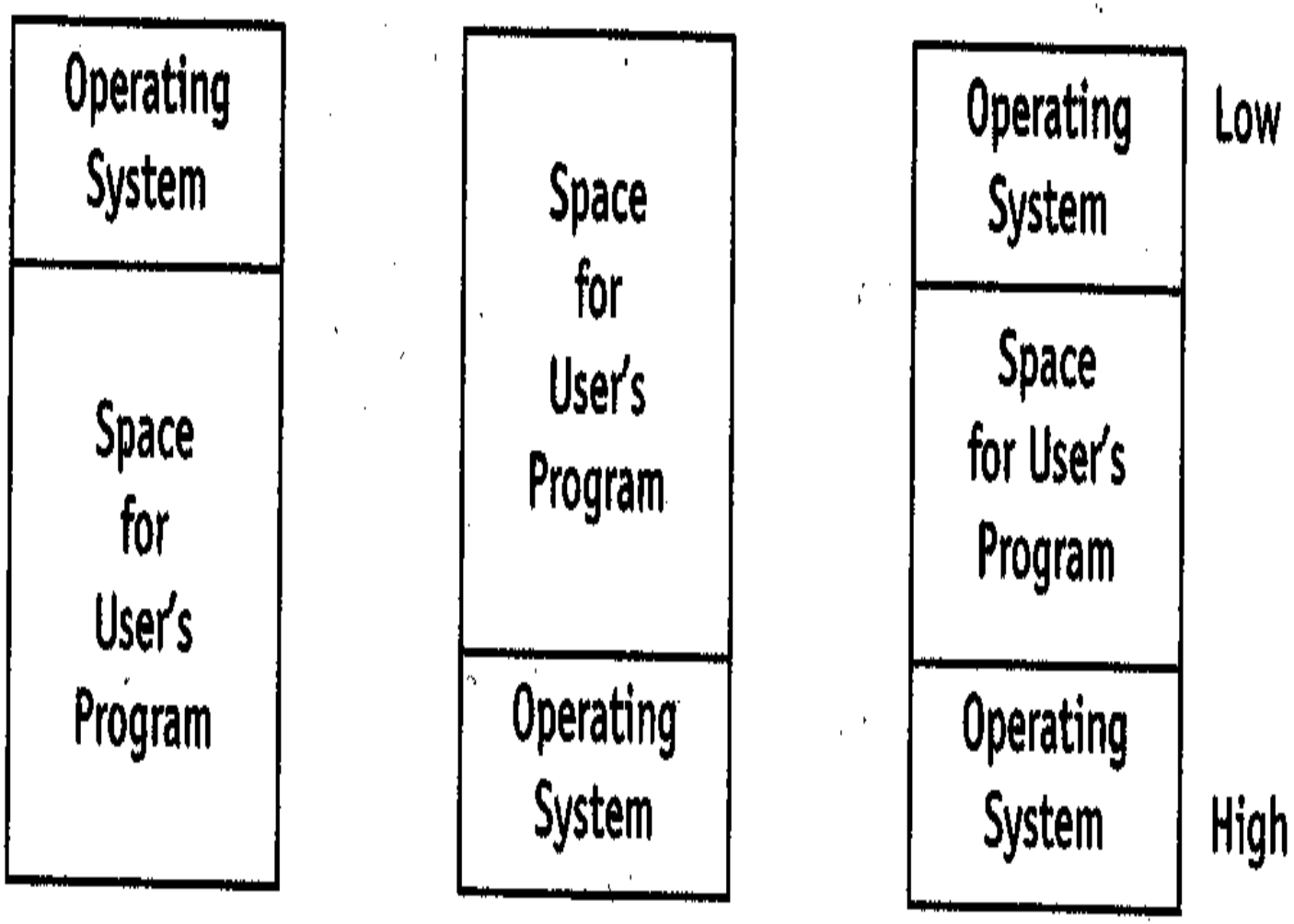
- *Input devices* provide main memory and/or the CPU with data from the outside world.
- *Output devices* provide information to the outside world from main memory or the CPU.
- Some devices perform both input and output (e.g., a disk drive).
- “I/O” refers to input, output, and input/output devices.

FIGURE 2.5


H1 CPU
Clock
pc
ac
sp
machine
language
instructions
and data
results
machine language
instructions
data
results
Input Device
Output Device
Main Memory

Location of OS in main memory

FIGURE 2.6



Load machine instruction

Opcode

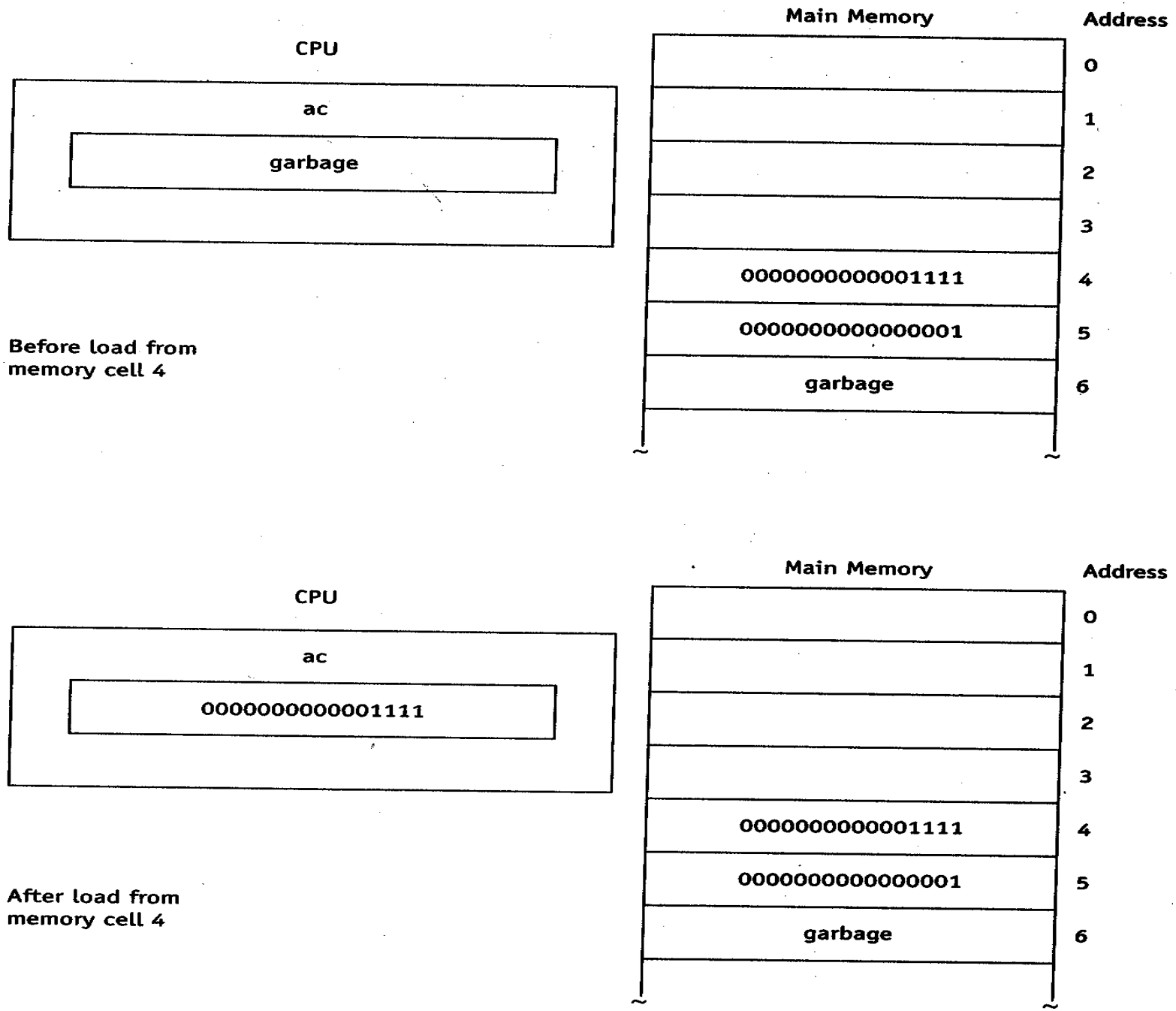
0000 000000000100

address

0004 in hex

This instruction loads the ac register from location 4 of main memory.

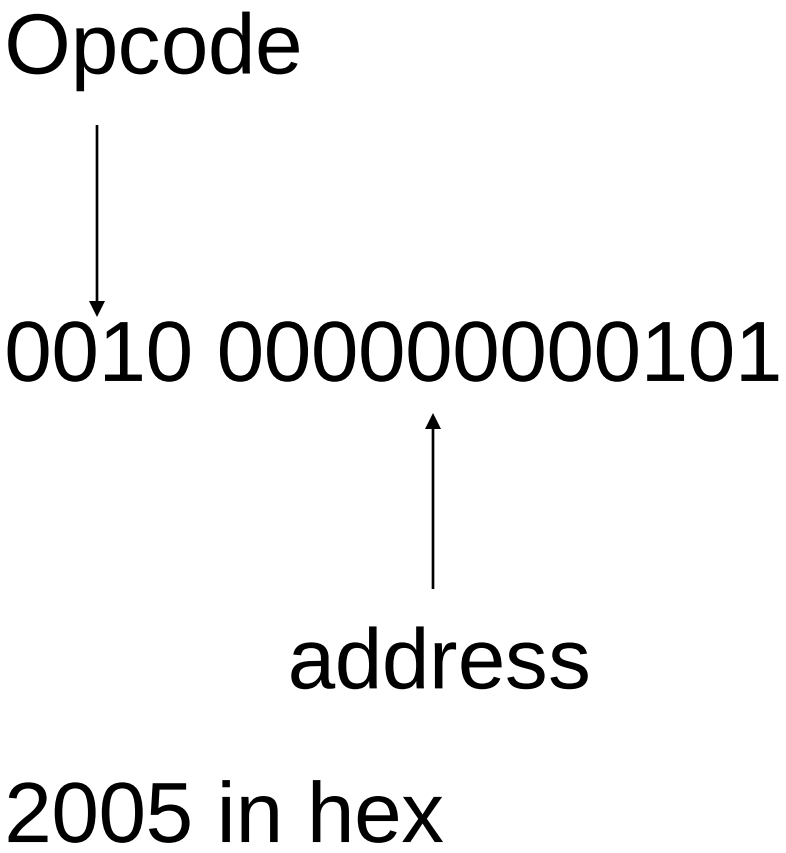
FIGURE 2.8


CPU
ac
garbage
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
garbage
6
Before load from
memory cell 4
CPU
ac
0000000000001111
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
garbage
6
After load from
memory cell 4

Some opcodes

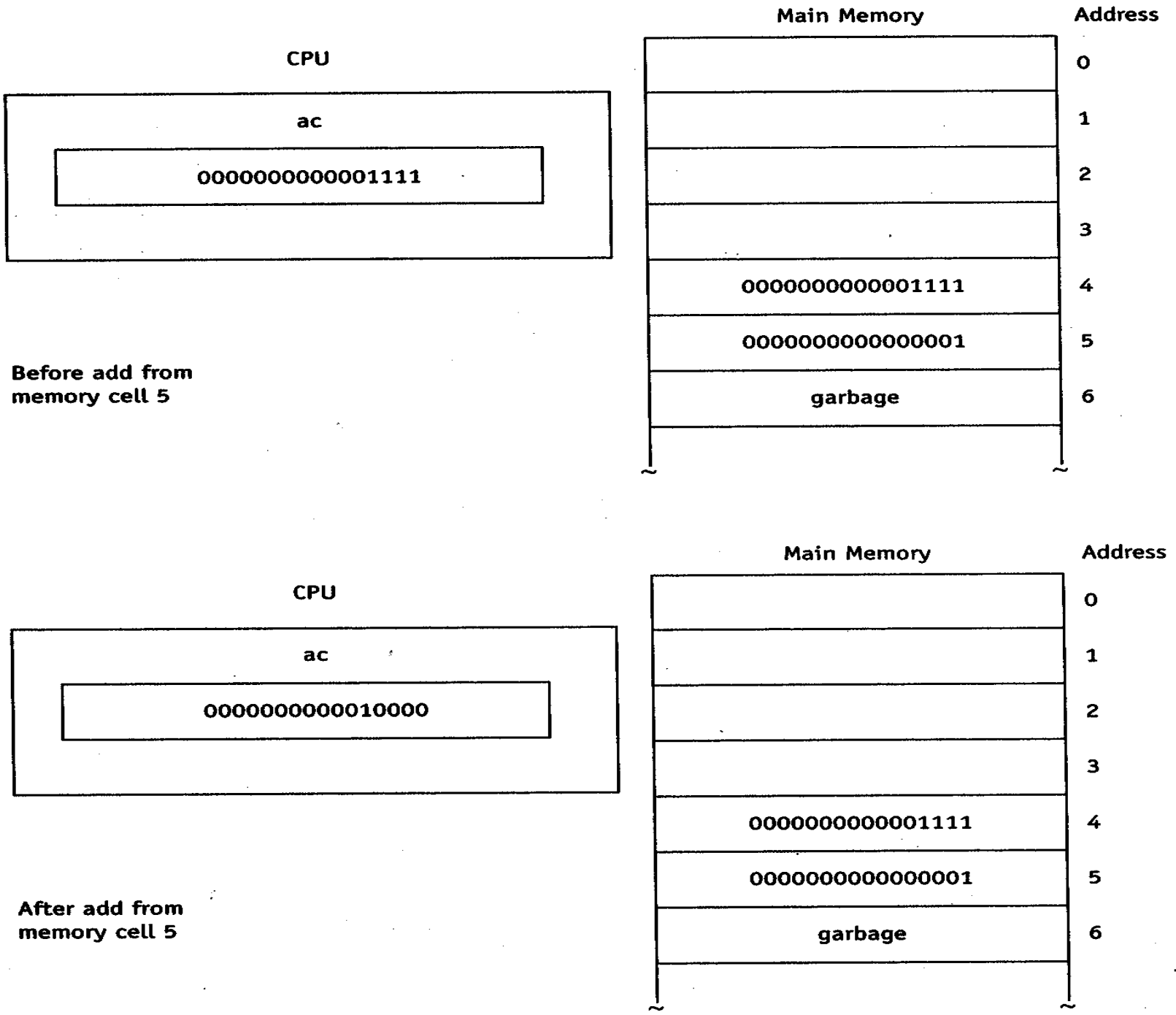
- 0000 load ac register from memory
- 0001 store ac register into memory
- 0010 add memory contents to ac register
- 1111111111111111 halt

Add machine instruction

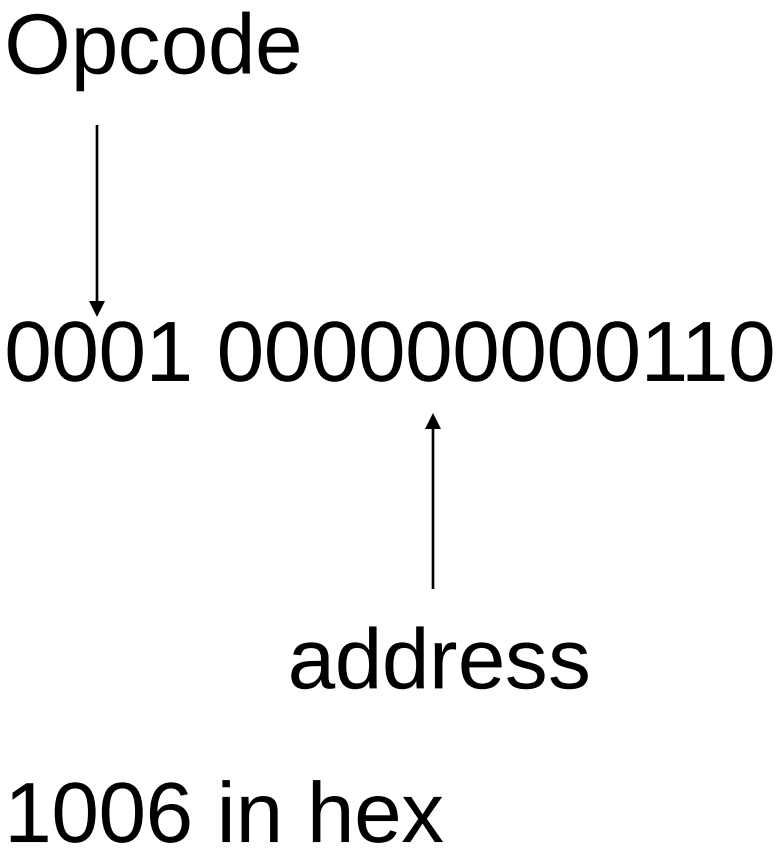


2005 in hex

FIGURE 2.9

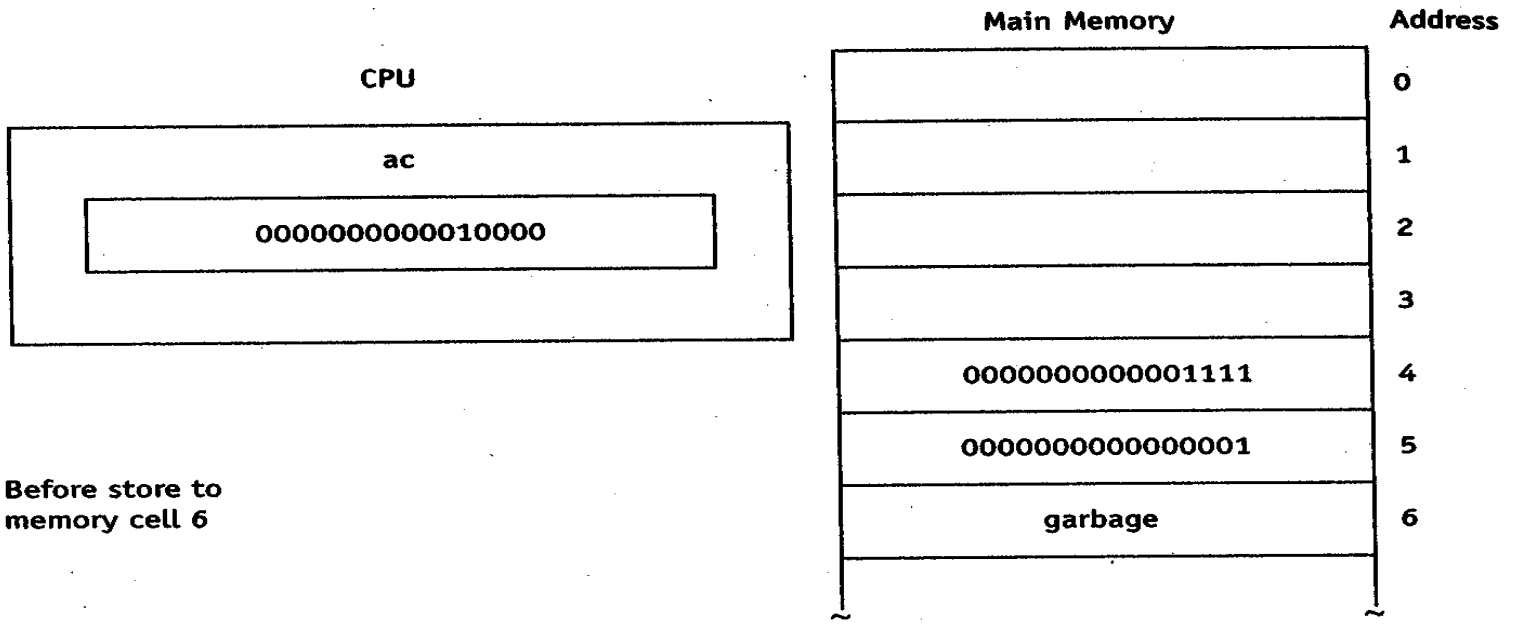

CPU
ac
0000000000001111
Before add from
memory cell 5
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
garbage
6
CPU
ac
0000000000010000
After add from
memory cell 5
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
garbage
6

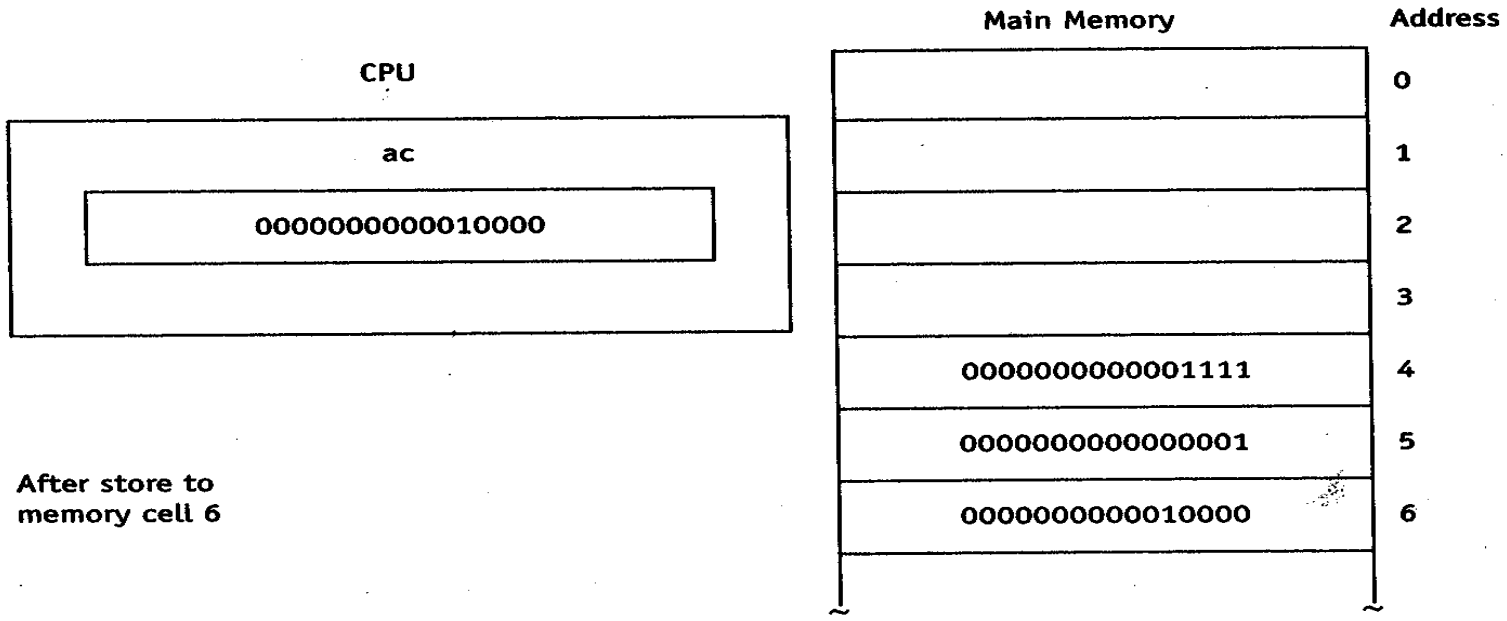
Store machine instruction



1006 in hex

FIGURE 2.10


CPU
ac
0000000000010000
Before store to
memory cell 6
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
garbage
6


CPU
ac
0000000000010000
After store to
memory cell 6
Main Memory
Address
0
1
2
3
0000000000001111
4
0000000000000001
5
0000000000010000
6

Halt machine instruction

Opcode (16 bits)

1111111111111111

FFFF in hex

Machine language program

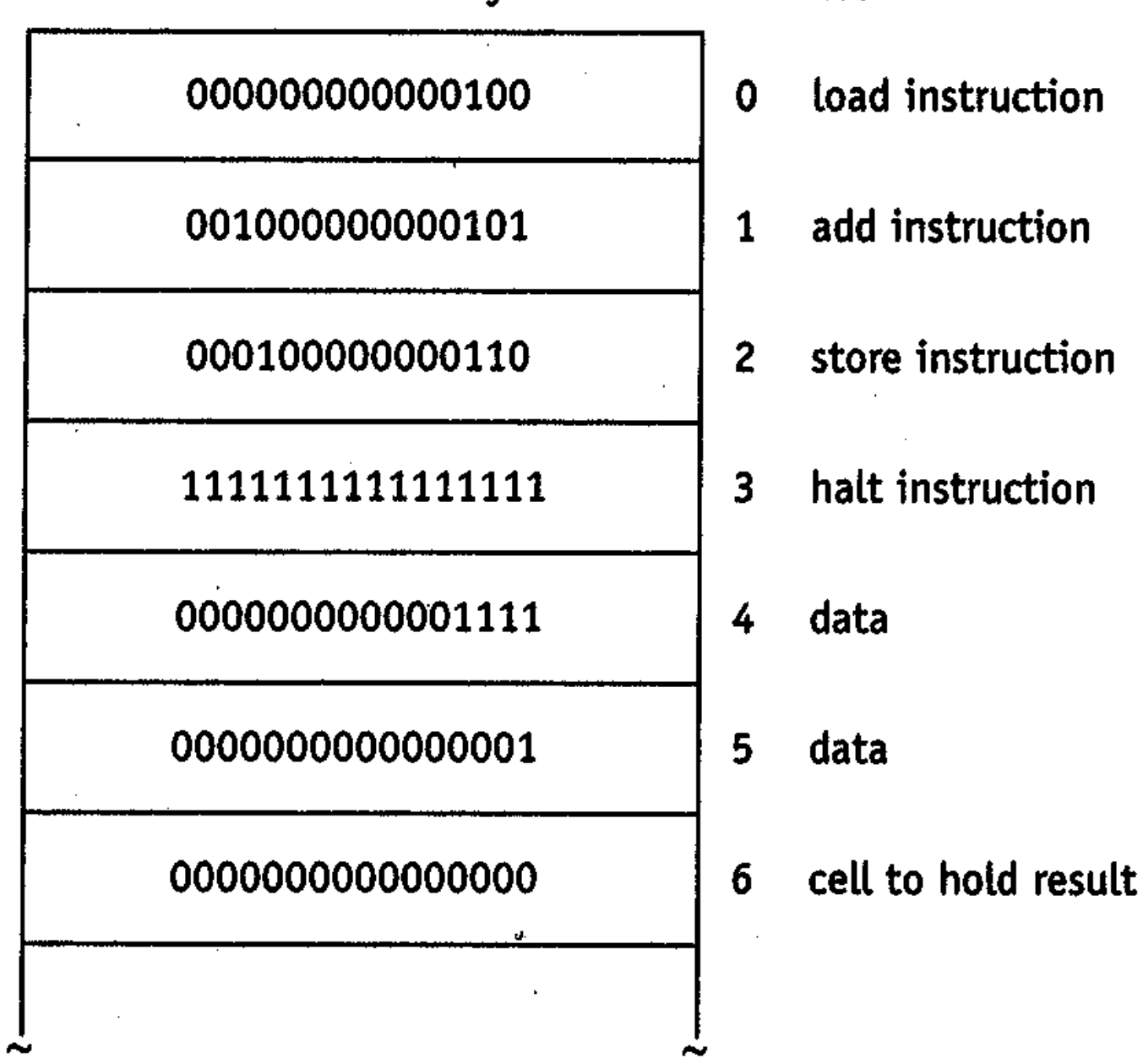
FIGURE 2.11

Address	Machine Instructions and Data in Hex	Description
0	0004	load instruction
1	2005	add instruction
2	1006	store instruction
3	FFFF	halt instruction
4	000F	data
5	0001	data
6	0000	cell to hold the result

Important reminder

- We use hex numbers as a shorthand representation of binary numbers.
- All the numbers in the computer are in binary.
- The next slide shows what our simple machine language program really looks like as it sits in memory.

FIGURE 2.12


Main Memory
Address
000000000000100
0 load instruction
001000000000101
1 add instruction
000100000000110
2 store instruction
111111111111111
3 halt instruction
000000000001111
4 data
000000000000001
5 data
000000000000000
6 cell to hold result

sim

- **sim** is a program that makes your computer act like H1.
- **sim** runs on DOS, Windows, Sun Sparc Solaris, X86 Linux, and Macintosh OS X.
- **sim** is better for our purposes than a real H1: With **sim** we can easily make changes to H1.
- **sim** has a powerful debugger.

Some commands for **sim**'s debugger

e (edit memory)
d (display memory)
t (trace)
r (register display)
f (file)
q (quit)

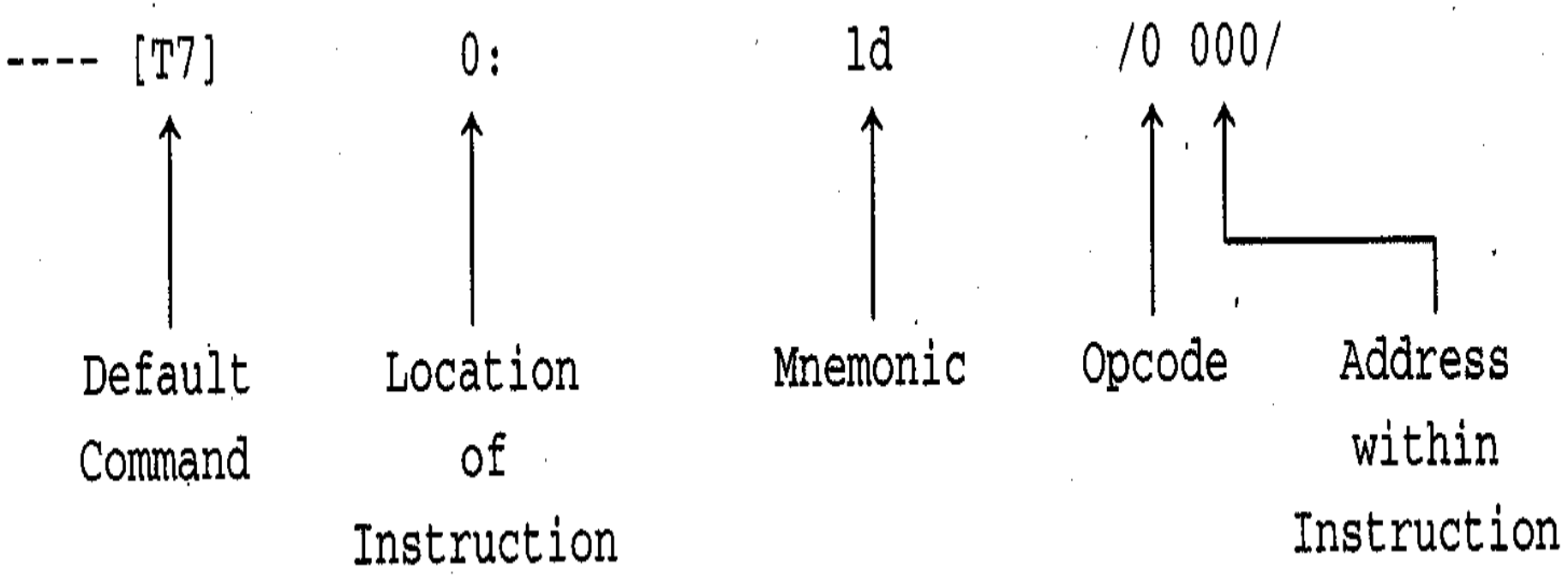
When a register changes its contents, **sim**'s debugger shows its before and after values.

```
ac=000F/0010
```

Mnemonics

- A *mnemonic* is an easy-to-remember representation of an opcode.
- ld mnemonic for the load opcode
- st mnemonic for the store opcode
- add mnemonic for the add opcode
- halt mnemonic for the halt opcode

Debugger's prompt provides a lot of information



The next slide shows how to use the debugger to enter and run our simple machine language program.

The bold type corresponds to user input.

FIGURE 2.13

```
C:\H1>sim
Simulator Version x.x
Enter machinecode file name and/or args, or hit ENTER to quit
none

Starting session.
Enter h or ? for help.
---- [T7] 0: ld   /0 000/ e0
  0: 0000/0004                          ←enter machine language program
  1: 0000/2005
  2: 0000/1006
  3: 0000/ffff
  4: 0000/000f
  5: 0000/0001
  6: 0000/0000
  7: 0000/                              ←hit ENTER to exit edit mode
---- [T7] 0: ld   /0 004/ d0            ←display memory from location 0
  0: 0004 2005 1006 FFFF 000F 0001 0000 0000     ........
  8: 0000 0000 0000 0000 0000 0000 0000 0000     ........
 10: 0000 0000 0000 0000 0000 0000 0000 0000     ........
 18: 0000 0000 0000 0000 0000 0000 0000 0000     ........
---- [T7] 0: ld   /0 004/              ←hit ENTER to trace program with T7
  0: ld   /0 004/ ac=0000/000F
  1: add  /2 005/ ac=000F/0010
  2: st   /1 006/ m[006]=0000/0010
  3: halt /FFFF /
Machine inst count =      4 (hex) =       4 (dec)
---- [T7] r*                          ←display all registers
     pc    = 0004      sp    = 0000      ac    = 0010
---- [T7] d6                          ←display memory from location 6
  6: 0010 0000 0000 0000 0000 0000 0000 0000     ........
  E: 0000 0000 0000 0000 0000 0000 0000 0000     ........
 16: 0000 0000 0000 0000 0000 0000 0000 0000     ........
 1E: 0000 0000 0000 0000 0000 0000 0000 0000     ........
---- [T7] f 0 6                       ←write locations 0 to 6 to a file
Enter file name.    [f.mac]
simple                                ←file name to use
Writing locations 0 - 6 to simple.mac
---- [T7] q                           ←quit sim
C:\H1>
```

Creating a log file

FIGURE 2.14

```
C:\H1>sim
Simulator Version x.x
Enter machinecode file name and/or args, or hit ENTER to quit.
none

Starting session.  Enter h or ? for help.
---- [T7] 0: ld   /0 000/ l        ←the letter "L"
Log file none.log is now on
---- [T7] 0: ld   /0 000/ e0
  0: 0000/0004

          .
          .
```

sim does not prompt for arguments if they are provided on the command line when **sim** is first invoked.

FIGURE 2.15

```
C:\H1>sim none
Simulator Version x.x

Starting session.  Enter h or ? for help.
---- [T7] 0: ld   /0 000/
.
.
.
```

Getting help on **sim**

Enter one of the following:

sim /h
sim –h
sim /?
sim -?

Avoid ‘?’ on Sun, Linux, and OS X

Getting help on **sim**'s debugger

Enter one of the following:

h
?

when the debugger is active.